Journal of Computational and Applied Mathematics 31 (1990) 105–118
North-Holland

Random number generation with the recursion $X_t = X_{t-3p} \oplus X_{t-3q}$ *

Masanori FUSHIMI
Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo, Bunkyo-ku, Tokyo 113, Japan

Received 16 December 1989

Abstract: A generalized feedback shift register (GFSR) algorithm proposed by Lewis and Payne (1973) uses a primitive trinomial to generate a sequence of pseudorandom numbers. We propose a similar algorithm which uses a primitive polynomial with many nonzero terms, but generates a number as fast as the original GFSR algorithm. Our sequence is guaranteed to be equidistributed in higher dimensions and to have a good autocorrelation property. Extensive statistical tests have been performed on the sequences generated by our algorithm and the results were quite satisfactory.

Keywords: M-sequence, GFSR algorithm, multidimensional uniformity, statistical tests.

1. Introduction

Marsaglia [17] pointed out about 20 years ago that the points in the unit n-cube generated by the Lehmer's linear congruential method lie in a relatively small number of parallel hyperplanes. All sequences generated with recursions of low degree over the finite field have the flaw of sparseness in n-space although the structure of sparseness may be different from generator to generator. After this finding, random numbers generated with recursive formulas of higher degree attracted the attention of many researchers. Since Tausworthe [23] and Lewis and Payne [16] published papers on the methods of generating pseudorandom numbers by linear recurrence modulo two, many papers have been written on these methods, and they have also been discussed in some monographs on statistical computing and simulation. There are some false or misleading statements in these papers and monographs, however, partly because Lewis and Payne's paper contains some errors which had not been recognized for a long time [10]. Some writers suspect the applicability of number sequences generated by linear recurrence modulo two; Marsaglia [18], in particular, states that these sequences should never be used.

We have developed a new method of generating pseudorandom numbers by linear recurrence modulo two. The method uses the recurrence formula

$$X_t = X_{t-3p} \oplus X_{t-3q} \tag{1}$$

* Part of this research was supported by Grant-in-Aids for Scientific Research of the Ministry of Education, Science and Culture of Japan under Grants (A) 60302035 (1985) and (C) 63580020 (1988).

instead of the formula

$$X_t = X_{t-p} \oplus X_{t-q} \tag{2}$$

proposed by Lewis and Payne, where $1 + z^q + z^p$ is a primitive polynomial of degree p over the Galois field GF(2), and the symbol $\oplus$ means bitwise addition modulo two. Since a random number is generated with only one exclusive-or operation, our method is as fast as, or even faster than, the linear congruential method. The crucial point in using this generator as well as other generators based on linear recurrences modulo two is the initialization procedure. The details of the procedure will be described in Section 4 and a FORTRAN program based on it is listed in the Appendix. The sequence $\langle X_t \rangle$ generated by (1) with this initialization procedure is free from the sparseness in n-space and has the period $T = 2^p - 1$. If the bit length l of X_t is an integral power of 2, e.g., $l = 32$, the sequence is guaranteed to be $\lfloor p/l \rfloor$-distributed, and the autocorrelation function $R(s)$ defined below has a value of $O(T^{-1})$ for $1 \leqslant s < T/l$:

$$R(s) = \frac{1}{T} \sum_{t=1}^{T} (X_t - \overline{X})(X_{t+s} - \overline{X}) 2^{-2l}, \tag{3}$$

$$\overline{X} = \frac{1}{T} \sum_{t=1}^{T} X_t \approx \tfrac{1}{2} \cdot 2^l. \tag{4}$$

The parameter p is normally taken to be greater than 100, so that the value T/l is quite large.

Using the primitive trinomial $1 + z^{32} + z^{521}$, we have performed extensive statistical tests on number sequences generated by our method. Tests performed include one of Marsaglia's "stringent tests" [18], OPSO (overlapping-pairs-sparse-occupancy) test, as well as many "standard" ones, and the results of all the tests were satisfactory. In the OPSO test, for example, Marsaglia reports that the statistic, which should be a standard normal, produced by his generator using the recurrence formula $X_t = X_{t-17} \oplus X_{t-5}$ assumed 10 successive values of 2895.9. Our result presented a striking contrast to his result: 400 deviates were produced among which one exceeded 3 in absolute value, and their distribution passed the Kolmogorov–Smirnov test.

2. Previous results

Let

$$f(D) = 1 + c_1 D + c_2 D^2 + \cdots + c_p D^p, \quad c_p = 1, \tag{5}$$

be a primitive polynomial of degree p over the Galois field GF(2), and $\langle a_t \rangle$ be the sequence of 0s and 1s generated by the linear recursion

$$a_t = c_1 a_{t-1} + c_2 a_{t-2} + \cdots + c_p a_{t-p} \pmod 2, \tag{6}$$

or equivalently, in the operator form,

$$f(D) a_t = 0 \pmod 2, \tag{7}$$

where D is understood to be the operator which decreases the subscript by 1. As the initial values

$(a_0, a_1, \ldots, a_{p-1})$ for the recursion (6) we can choose any values other than $(0, 0, \ldots, 0)$, and the period of the sequence $\langle a_t \rangle$ is

$$T = 2^p - 1. \tag{8}$$

The sequence $\langle a_t \rangle$ is called by various names such as an M-sequence, a pseudonoise (PN) sequence, a feedback shift register sequence, and its properties are well known [11]. The function $f(D)$ is called the characteristic polynomial of the M-sequence $\langle a_t \rangle$.

2.1. Tausworthe sequence

Using an M-sequence $\langle a_t \rangle$, Tausworthe proposed to form an l-bit binary fractional sequence $\langle x_t \rangle$ defined by

$$x_t = 0.a_{\sigma t} a_{\sigma t+1} \ldots a_{\sigma t+l-1} \text{ (base 2)}, \tag{9}$$

where σ is a positive constant which satisfies the following conditions:

$$\sigma \geqslant l, \tag{10}$$

$$\gcd(\sigma, T) = 1. \tag{11}$$

He has shown that the sequence $\langle x_t \rangle$ has virtually the same mean and variance as the uniform distribution in the interval [0, 1], has virtually no serial correlation for lags up to $(T - l)/\sigma$, and is k-distributed for $1 \leqslant k \leqslant \lfloor p/\sigma \rfloor$. Here the term "$k$-distribution" means the property that every k-tuple of l-bit numbers appears 2^{p-kl} times over the full period except the all-zero tuple which appears one time less.

2.2. GFSR sequence

Lewis and Payne [16] introduced an apparently different type of generator, the generalized feedback shift register (GFSR) generator, by which numbers are formed by phase-shifted elements along an M-sequence based on a primitive trinomial $1 + z^q + z^p$:

$$y_t = 0.a_t a_{t+\tau} \ldots a_{t+(l-1)\tau} \text{ (base 2)}. \tag{12}$$

This method is fast because the integer sequence $\langle Y_t \rangle = \langle 2^l y_t \rangle$ can be generated by the recursion

$$Y_t = Y_{t-p} \oplus Y_{t-q}. \tag{13}$$

In order to generate the sequence $\langle Y_t \rangle$, using this recursion, we must give initial values $Y_1, Y_2, \ldots, Y_p$. Let A be the $p \times l$ 0-1 matrix consisting of the elements of the M-sequence $\langle a_t \rangle$ contained in these initial values. Lewis and Payne proposed to put all 1s in the first column of A, i.e., to choose $a_1 = a_2 = \cdots = a_p = 1$, and suggested to choose the phase shift between adjacent columns τ as $100p$ and to generate the elements in the succeeding columns by repeatedly applying the basic recurrence formula. They also recommended to discard some $5000p$ Ys before using the sequence in order to diminish the effect of the initial column of 1s. Thus their initializing procedure is extremely time consuming.

As to the multidimensional distribution property of the sequence, they state as follows. The GFSR algorithm produces multidimensional pseudorandom numbers. ...(1) It is only necessary that the seed matrix A has linearly independent columns. ...(2) Linear independence is

guaranteed if the maximum delay measured from the leftmost column is less than the full period $2^p - 1$ (if the "constant delay" between each column is relatively prime to $2^p - 1$, the maximum delay can exceed the full period).

It was pointed out in [10] that the statements (1) and (2) are false: the linear independence of the columns of the seed matrix A is merely the condition for one-dimensional uniformity; the stated condition on the delay between columns does not guarantee the independence of columns. A necessary and sufficient condition for multidimensional uniformity of the GFSR sequence was found by Fushimi and Tezuka [10].

Theorem 1. *The l-bit GFSR sequence based on the primitive polynomial of degree p is k-distributed if and only if the kl elements in the first k rows of the seed matrix A are linearly independent in the sense that the $kl \times p$ matrix which expresses these kl elements as a linear transformation of some (typically the first) column vector in the seed matrix is nonsingular.*

After Lewis and Payne's algorithm was published, several authors proposed methods for improving the initialization procedure [1,4,14,20], but they did not recognize that the Lewis and Payne's condition (1) is false so that these methods do not necessarily generate good pseudorandom numbers. Based on Theorem 1, Fushimi and Tezuka [10] proposed a fast initialization procedure which is guaranteed to generate numbers with multidimensional uniformity. Another initialization procedure which is more convenient can be devised, however, if we recognize the relationship between Tausworthe and GFSR sequences which will be shown below.

2.3. Equivalence relation between Tausworthe and GFSR sequences

Although the Tausworthe sequence (9) and the GFSR sequence (12) look apparently different, there is a close relationship between them. In fact, Fushimi [6] has shown the following equivalence relation between the class of Tausworthe sequences without the condition (10) and that of GFSR sequences dropping the condition that $\langle a_t \rangle$ is generated by a trinomial. The key concept in the following is the "proper decimation" of the M-sequence, in which "decimation" means selecting every nth term of the M-sequence, and "proper" means that n is relatively prime to the period $2^p - 1$. It is known [11] that, if $\langle a_t \rangle$ is an M-sequence whose characteristic polynomial $f(D)$ is of degree p, then a properly decimated sequence $\langle a_{nt};\ t = 1, 2, 3, \ldots \rangle$ is also an M-sequence whose characteristic polynomial, say $f_n(D)$, is of degree p, and that $f_n(D) = f(D)$ if and only if n is an integral power of 2. Let n^{-1} denote the inverse of n with respect to the multiplication modulo $2^p - 1$. In the following, we use the notation $\langle x_t(f;\ \sigma) \rangle$ and $\langle y_t(f;\ \tau) \rangle$ to specify the primitive polynomials and the parameters used to construct $\langle x_t \rangle$ and $\langle y_t \rangle$; we write $\langle x_t \rangle \cong \langle y_t \rangle$ when the two sequences $\langle x_t \rangle$ and $\langle y_t \rangle$ are equivalent in the sense they are the same sequence except possibly the location of the starting point.

Theorem 2. *If f is a primitive polynomial of degree p, and σ and τ are relatively prime to $2^p - 1$, then the following equivalence relations hold:*

$$\langle x_t(f;\ \sigma) \rangle \cong \langle y_t(f_\sigma;\ \sigma^{-1}) \rangle, \tag{14a}$$

$$\langle y_t(f;\ \tau) \rangle \cong \langle x_t(f_\tau;\ \tau^{-1}) \rangle. \tag{14b}$$

Using this theorem, we can derive the following fast initialization procedure for the GFSR generator [6]. We will confine ourselves to the case where the bit length l is an integral power of 2, but the other cases can be treated with a little modification. If we set $\tau = 2^p/l$, then we have $f_\tau = f$ and $\tau^{-1} = l$ so that the GFSR sequence $\langle y_t(f;\ \tau)\rangle$ is equivalent to the Tausworthe sequence $\langle x_t(f;\ l)\rangle$. Thus we can use p initial values of the Tausworthe sequence as initial values for the GFSR sequence. Since $\langle x_t(f;\ l);\ 1 \leqslant t \leqslant p\rangle$ is constructed by initial lp elements of the M-sequence, it can be generated very fast; in fact, some 250 times as fast as Lewis and Payne's algorithm if $l = 32$ and $p = 521$.

Our procedure has two other advantages over Lewis and Payne's algorithm. First, the sequence $\langle y_t\rangle$ initialized by our procedure is guaranteed to be k-distributed for $1 \leqslant k \leqslant \lfloor p/l \rfloor$. Second, our sequence has a good correlation property. The value of the autocorrelation function of any GFSR sequence is almost zero for lags up to τ provided that $l\tau \leqslant 2^p - 1$. Thus it is desirable to select τ as large as possible. Our τ is the largest possible, while Lewis and Payne's τ, i.e., $100p$, is very small and not enough for large scale Monte Carlo simulations. Arvillias and Maritsas [1] proposed an efficient method for selecting τ almost as large as possible , but their method is applicable only when the characteristic polynomial is a trinomial $1 + D^q + D^p$ and q is an integral power of 2. Moreover the k-distributivity is not guaranteed by their method unless another criterion which is introduced by Fushimi and Tezuka in [10] is fulfilled.

3. The generator $X_t = X_{t-3p} \oplus X_{t-3q}$

The GFSR sequence as well as the Tausworthe sequence can be constructed using any M-sequence whether the characteristic polynomial is trinomial or not; in fact, Theorems 1 and 2 hold for any primitive polynomial f. Considering the speed of generation, however, primitive trinomials have been used almost exclusively by many researchers. The rationale of using the recursion (1) rather than (2) is to construct a GFSR sequence using a primitive polynomial with many nonzero terms and to generate it very fast.

Let $f(D)$ be a primitive polynomial of degree p defined by (5), $Q(D)$ be any polynomial, and $F(D)$ be their product:

$$F(D) = Q(D)f(D) = 1 + C_1D + C_2D^2 + \cdots + C_sD^s, \quad C_s = 1. \tag{15}$$

Then the M-sequence $\langle a_t\rangle$ defined by (7) also satisfies the recursion $F(D)a_t = 0 \pmod 2$, or equivalently

$$a_t = C_1a_{t-1} + C_2a_{t-2} + \cdots + C_sa_{t-s} \pmod 2. \tag{16}$$

If the number of nonzero terms in (16) is less than that in (6), then the sequence $\langle a_t\rangle$ can be generated faster by using the recursion (16) rather than (6). In particular, if we can find $Q(D)$ such that $F(D)$ becomes a trinomial, then we can generate a_t with only one exclusive-or operation. It is very difficult in general, however, to find such $Q(D)$ for an arbitrarily given $f(D)$. Going in the reverse way, therefore, we will show that for a suitably chosen (reducible) trinomial $F(D)$ there exists a primitive polynomial which divides $F(D)$.

Let $\langle b_t\rangle = \langle a_{3t}\rangle$ be a proper decimation of the M-sequence $\langle a_t\rangle$. The decimation is proper, i.e., 3 is relatively prime to the period $T = 2^p - 1$, if and only if p is an odd integer, so that we put

$$p = 2m + 1. \tag{17}$$

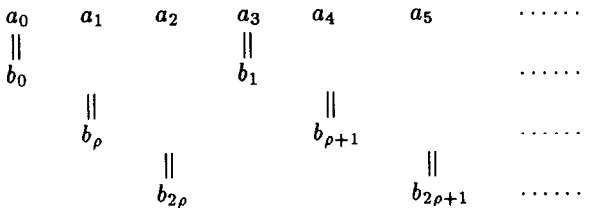

Fig. 1. The relationship between two M-sequences $\langle a_t \rangle$ and $\langle b_t \rangle = \langle a_{3t} \rangle$.

Let ρ be defined by

$$\rho = \tfrac{1}{3}(2T+1) = 4^m + 4^{m-1} + \cdots + 4 + 1. \tag{18}$$

It is easy to see that the following relations hold:

$$3\rho = 1 \pmod{T}, \tag{19}$$

$$\langle a_t \rangle = \langle b_{\rho t} \rangle. \tag{20}$$

The relationship between two M-sequences $\langle a_t \rangle$ and $\langle b_t \rangle$ is shown in Fig. 1. Thus we can consider that the sequence $\langle b_t \rangle$ is given at first and $\langle a_t \rangle$ is defined by (20). As the characteristic polynomial of $\langle b_t \rangle$ we choose the trinomial

$$g(D) = 1 + D^q + D^p, \quad p > q. \tag{21}$$

The characteristic polynomial $f(D)$ of $\langle a_t \rangle$ is different from $g(D)$ [11], and usually it has many nonzero terms. On the other hand, $\langle a_t \rangle$ satisfies the relation:

$$\begin{aligned} a_t + a_{t-3p} + a_{t-3q} &= b_{\rho t} + b_{\rho(t-3p)} + b_{\rho(t-3q)} \\ &= b_{\rho t} + b_{\rho t - p} + b_{\rho t - q} = 0 \pmod 2, \end{aligned} \tag{22}$$

so that it can be generated by the recursion

$$a_t = a_{t-3p} + a_{t-3q} \pmod 2, \tag{23}$$

which means the trinomial

$$F(D) = g(D^3) = 1 + D^{3q} + D^{3p} \tag{24}$$

is divisible by a primitive polynomial $f(D)$.

Using the sequence $\langle a_t \rangle$, we construct an l-bit binary sequence $\langle x_t(f;\ l) \rangle$ as follows [1]:

$$x_t = 0.a_{lt}a_{lt+1}\ldots a_{lt+l-1} \text{ (base 2)}. \tag{25}$$

Since we have assumed l is an integral power of 2, it follows from Theorem 2 that $\langle x_t(f;\ l) \rangle \cong \langle y_t(f;\ \tau) \rangle$, where $\tau = 2^p/l$. Thus the integer sequence $\langle X_t \rangle = \langle 2^l x_t(f;\ l) \rangle$ can be generated by the recursion

$$X_t = X_{t-3p} \oplus X_{t-3q}. \tag{26}$$

[1] For simplicity of description we assume l is an integral power of 2, but the modification for other cases is quite easy.

4. Initializing the generator

In order to generate the sequence $\langle X_t \rangle$ using the recursion (26), we must give initial values X_t $(0 \leqslant t \leqslant 3p-1)$. These values are determined by specifying the first $3lp$ elements a_t $(0 \leqslant t \leqslant 3lp-1)$ of the M-sequence $\langle a_t \rangle$, of which the last $3(l-1)p$ can be generated by the recursion (23) if we give a_t $(0 \leqslant t \leqslant 3p-1)$. The values of these $3p$ elements cannot be specified freely, however, because $\langle a_t \rangle$ is an M-sequence whose characteristic polynomial $f(D)$ is a primitive polynomial of degree p. Since we have the relationship

$$\{a_t;\ 0 \leqslant t \leqslant 3p-1\} = \{b_t;\ 0 \leqslant t \leqslant p-1\} \cup \{b_{\rho+t};\ 0 \leqslant t \leqslant p-1\} \cup \{b_{2\rho+t};\ 0 \leqslant t \leqslant p-1\}, \tag{27}$$

which can easily be seen from Fig. 1, we can freely give the values of the p elements in the last set on the right-hand side of (27), and compute the values of the remaining $2p$ elements using the following identities:

$$S_\rho(D) = S_{2\rho}(D)D^\rho \pmod{g(D)}, \tag{28}$$

$$S_0(D) = S_\rho(D)D^\rho \pmod{g(D)}. \tag{29}$$

Here $S_i(D)$ is a polynomial in D of degree $p-1$ defined by

$$S_i(D) = \sum_{k=0}^{q-1} b_{i+p-q+k}D^k + \sum_{k=q}^{p-1} b_{i-q+k}D^k. \tag{30}$$

Thus the coefficients of $S_{2\rho}(D)$, $S_\rho(D)$ and $S_0(D)$ are the elements of the sets $\{b_{2\rho+t};\ 0 \leqslant t \leqslant p-1\}$, $\{b_{\rho+t};\ 0 \leqslant t \leqslant p-1\}$ and $\{b_t;\ 0 \leqslant t \leqslant p-1\}$, respectively.

The validity of the formulas (28) and (29) can be shown in the following way. We will first derive the relationship between $S_i(D)$ and $S_{i-1}(D)$:

$$S_{i-1}(D) = b_{i-1+p-q} + \sum_{k=1}^{q-1} b_{i-1+p-q+k}D^k + b_{i-1}D^q + \sum_{k=q+1}^{p-1} b_{i-1-q+k}D^k. \tag{31}$$

Since we have

$$D^p + D^q = 1 \pmod{g(D)}, \tag{32}$$

$$b_{i-1} + b_{i-1+p-q} = b_{i-1+p} \pmod 2, \tag{33}$$

the relationship

$$b_{i-1+p-q} + b_{i-1}D^q = b_{i-1+p}D^q + b_{i-1+p-q}D^p \pmod{g(D)} \tag{34}$$

holds and $S_{i-1}(D)$ can be written as

$$S_{i-1}(D) = \sum_{k=1}^{q} b_{i-1+p-1+k}D^k + \sum_{k=q+1}^{p} b_{i-1-q+k}D^k = S_i(D)D \pmod{g(D)}. \tag{35}$$

Using this recurrence relation j times, we obtain

$$S_{i-j}(D) = S_i(D)D^j \pmod{g(D)}. \tag{36}$$

Identities (28) and (29) are obtained by putting $(i,\ j) = (2\rho,\ \rho)$ and $(\rho,\ p)$ in (36), respectively.

If we use recursions (28) and (29), we must compute $D^{\rho} \pmod{g(D)}$. Although ρ defined by (18) is astronomically large, the computation can be done very fast if we use the following recursion. Let $h_k(D)$ be defined by

$$h_k(D) = D^{4^k+4^{k-1}+\cdots+4+1} \pmod{g(D)};$$

then our goal is to obtain $h_m(D)$. It is easy to see that the following recursion for $h_k(D)$ holds:

$$h_{k+1}(D) = \{h_k(D)\}^4 D = \left[\{h_k(D)\}^2 \pmod{g(D)}\right]^2 D \pmod{g(D)}. \tag{37}$$

Starting from $h_0(D) = D$, we use this recursion for $k = 0, 1, \ldots, m-1$ to obtain $h_m(D)$. Since

$$\left(\sum_{i=0}^{p-1} \beta_i D^i\right)^2 = \sum_{i=0}^{p-1} \beta_i D^{2i}$$

over GF(2), the squaring operations in (37) are quite easy.

A FORTRAN subroutine INTLZ(IX) in the Appendix is a program for initializing the generator using the techniques described in this section. It uses a primitive trinomial $g(D) = 1 + D^q + D^p$ with $p = 521$ and $q = 32$ but other trinomials can be used as well by changing the parameter values p and q (P and Q in the program list). The program is for IBM 370 or equivalent computers, which use 32 bits for representing an integer and use 2's compliment to represent a negative integer, but modifications for use with other computers are quite easy. The argument IX, which must be given a positive integer before calling, is used as the initial value for the multiplicative congruential generator (line 25) that determines the values of b_{2p+t} $(0 \leqslant t \leqslant p-1)$. Another subroutine RND (IRND, FRND) returns, whenever called, an integer random number IRND and a fractional random number FRND which correspond to X_t and x_t, respectively. It is not difficult to modify these programs so that they may run faster if we utilize techniques used in [25], but we will not go into details here.

The function IEOR on line 96 performs an exclusive-or operation. Since it is not a standard function in FORTRAN, it must be changed appropriately depending on the computer one actually uses.

5. Statistical tests

We have performed extensive statistical tests on the number sequences generated by the program in the Appendix. Since the details of the results of these tests were reported elsewhere [7,9], we will describe only the summary here.

We gave the following eight values to the argument IX: 1985, 9158, 9851, 8915, 8519, 5891, 5198, 1589; and performed the seven kinds of statistical tests given below for about one hundred million (10^8) random numbers generated by each of the above eight values:

(1) equidistribution test (one-dimensional frequency test); (2) serial test (two-dimensional frequency test); (3) poker test; (4) runs-up-and-down test; (5) gap test; (6) collision test; (7) OPSO test.

Details of the tests (1)–(6) are described in [15]. The OPSO test was proposed by Marsaglia [18] as one of the "stringent tests". In each test, we used s random numbers to compute a chi-square

Table 1
An example of the equidistribution test; IX = 1985

no.	K_{100}^{+}	K_{100}^{-}	no.	K_{100}^{+}	K_{100}^{-}	no.	K_{100}^{+}	K_{100}^{-}	no.	K_{100}^{+}	K_{100}^{-}
1	0.707	0.241	26	0.153	1.078	51	0.662	0.524	76	0.876	0.098
2	0.190	0.834	27	0.661	0.553	52	0.412	1.132	77	0.667	0.527
3	0.708	0.135	28	0.302	0.662	53	0.755	0.392	78	0.429	0.770
4	0.583	0.400	29	0.419	0.483	54	0.496	1.053	79	0.320	0.830
5	0.733	0.370	30	0.836	0.363	55	0.868	0.376	80	0.547	0.311
6	0.304	0.766	31	0.719	0.646	56	0.528	0.086	81	0.388	0.519
7	0.937	0.647	32	0.850	0.183	57	0.060	1.012	82	0.302	0.756
8	0.519	0.752	33	1.221	0.205	58	0.298	0.927	83	0.768	0.679
9	0.962	0.369	34	0.908	0.378	59	0.703	0.715	84	0.437	0.950
10	0.943	0.855	35	0.258	0.846	60	0.162	0.494	85	0.598	0.467
11	0.863	0.748	36	0.593	0.437	61*	0.041	1.579	86	0.944	0.416
12	0.736	0.375	37	0.476	0.466	62	0.755	0.334	87	0.266	0.760
13	0.294	1.237	38	0.454	0.900	63	0.897	0.751	88	0.357	0.869
14	0.641	0.562	39	0.937	0.565	64	0.579	0.427	89	0.802	0.458
15	0.593	0.753	40	0.662	0.473	65	0.656	0.554	90	0.310	0.730
16	0.563	0.355	41	0.168	1.061	66	0.263	0.958	91	0.590	0.495
17	0.783	0.224	42*	0.168	1.415	67	0.453	0.894	92	0.855	0.489
18	0.281	0.661	43*	0.096	1.358	68	0.755	0.460	93	0.472	0.386
19	0.622	0.466	44	0.553	1.113	69	1.283	0.519	94	0.267	0.904
20	0.195	0.711	45	0.429	0.933	70	0.967	0.300	95	0.843	0.455
21	0.256	0.641	46	1.104	0.143	71	0.385	0.541	96	0.393	0.481
22	1.047	0.058	47	0.501	0.946	72	0.818	0.383	97	0.370	0.695
23	0.260	0.699	48	0.444	0.284	73	0.482	0.988	98	0.595	0.256
24	0.390	0.566	49	0.130	1.297	74	0.516	0.365	99	0.717	0.513
25	0.600	0.537	50	0.305	0.873	75	0.841	0.672	100	0.554	0.554

SUMMARY: $K_{100}^{+} = 1.286$ $K_{100}^{-} = 0.189$

goodness-of-fit statistic χ_0^2, s being different from test to test. For 100 chi-square statistics thus computed we calculated Kolmogorov–Smirnov (K–S) statistics K_{100}^{+}, K_{100}^{-} and K_{100} [15].

In the equidistribution test, we generated $s = 1000$ integer random numbers $\lfloor 10x_t \rfloor$ and counted their frequency distribution f_i $(0 \leqslant i \leqslant 9)$ to compute a chi-square statistic with 9 degrees of freedom. Thus we obtained 1000 pairs of K–S statistics $(K_{100}^{+}, K_{100}^{-})$ for each initial value IX. Table 1 lists the first 100 pairs of K–S statistics obtained in this way for IX = 1985. Asterisks (*) indicate the blocks for which K_{100} exceeded the upper 5% point, i.e., 1.340. If the sequence $\langle x_t \rangle$ is ideally random, the number of asterisks will be distributed according to the binomial distribution B(100, 0.05), whose mean is 5 and standard deviation is 2.18. Since the number of asterisks in Table 1 is 3, our sequence is acceptable in this respect.

The bottom line (SUMMARY:) describes the K–S statistics for testing the distribution of 100 values of $K_{100} = \max\{K_{100}^{+}, K_{100}^{-}\}$ against the theoretical distribution of K_{100}, and we see that the maximum of these two values is below 1.340 and our sequence is also satisfactory from this point of view.

It is impossible to list in limited space the similar results for all other parts of the sequence we have tested, but those results were quite satisfactory and no significant abnormality of the sequence was observed. We also omit the description of the results of tests (2)–(5), but they were satisfactory too.

The collision test [15] has been designed to detect the nonrandomness of the points in higher dimensions generated by pseudorandom number generators. Let us consider a 10-dimensional unit-cube which is divided into $m = 4^{10}$ cells (cubes) with equal volumes. In this unit-cube we distribute, one after another, $n = 2^{14}$ points whose coordinates are determined by 10 successive numbers in the sequence $\langle x_t \rangle$. If a point falls into a cell that already contains at least one point, we say that a "collision" has occurred, and we count the number, say K, of collisions.

Table 2
An example of the collision test; IX = 1589

$K \leq$	101	108	119	126	134	145	153	999
PROBABILITY	0.009	0.034	0.201	0.232	0.266	0.204	0.043	0.011
CUM. PROB.	0.009	0.043	0.244	0.476	0.742	0.946	0.989	1.000
1	1	0	3	3	6	6	1	0
2	1	0	5	8	4	2	0	1
3	0	2	4	2	6	5	0	1
4	0	3	2	5	8	2	0	0
5	0	2	3	5	4	4	2	0
SUM 1	2	7	17	23	28	19	3	1
SUM 2			26	23	28			23
6	1	2	0	4	8	4	1	0
7	0	2	3	5	3	5	2	0
8	0	2	2	4	1	10	0	1
9	0	1	8	1	4	6	0	0
10	0	0	3	3	7	6	1	0
SUM 1	1	7	16	17	23	31	4	1
SUM 2			24	17	23			36
11	0	1	1	4	6	5	3	0
12	0	0	5	5	5	4	1	0
13	0	0	5	4	3	4	4	0
14	1	0	3	4	6	5	1	0
15	0	1	3	7	4	4	1	0
SUM 1	1	2	17	24	24	22	10	0
SUM 2			20	24	24			32
16	0	1	6	2	6	5	0	0
17	1	1	3	3	10	1	1	0
18	0	0	2	6	4	6	1	1
19	0	1	8	3	2	6	0	0
20	0	1	4	3	5	3	2	0
SUM 1	1	4	23	17	27	23	4	1
SUM 2			28	17	27			28
21	0	2	2	1	7	6	1	1
22	0	1	3	4	4	6	2	0
23	0	0	6	3	4	6	1	0
24	0	1	2	5	6	5	1	0
25	0	0	3	5	6	3	2	1
SUM 1	0	4	16	18	27	26	7	2
SUM 2			20	18	27			35
26	0	1	7	7	3	1	1	0
27	0	2	4	3	7	3	1	0
28	0	0	6	5	7	1	1	0
29	0	2	2	2	5	5	3	1
30	0	1	3	8	3	4	0	1
SUM 1	0	6	22	25	25	14	6	2
SUM 2			28	25	25			22

Table 3
Summary of collision tests

IX			χ_0^2	$(\phi = 3)$		
1985	4.099	5.703	3.367	4.372	9.213*	1.197
5198	1.433	8.291*	6.924	2.289	7.297	5.027
8519	2.278	0.784	0.841	0.674	4.391	4.055
9851	0.935	1.285	3.171	0.400	0.656	1.435
9158	2.006	2.084	4.403	1.953	1.572	1.215
1589	0.484	6.183	2.565	2.382	5.246	1.327
5891	6.865	0.367	0.516	0.185	1.598	2.725
8915	0.810	10.525*	7.648	5.885	0.821	0.336

Repeating the above process 20 times, we obtain the frequency distribution of K, an example of which is given on line 1 in Table 2: K assumed a value $\leqslant 101$ once, values between 109 and 119 inclusive 3 times, etc. Since these frequencies are too small to compute a chi-square goodness-of-fit statistic, we repeated the whole process 5 times (lines 1–5) and summed the frequencies in each column to get the frequency distribution (2, 7, 17, ...) on the line labeled "SUM 1". Some of these frequencies are still too small, and we pool them into 4 classes, i.e., $K \leqslant 119$, $119 < K \leqslant 126$, $126 < K \leqslant 134$, and $K > 134$, to get the frequencies (26,23,28,23) on the line "SUM 2". For this frequency distribution we computed a chi-square statistic with 3 degrees of freedom and obtained the value 0.484 on the line IX = 1589 in Table 3. (Note that $10 \times 2^{14} \times 20 \times 5 = 16\,384\,000$ random numbers were used to compute this value.) The other values in Table 3 were computed similarly.

The upper 5 and 1 percentage points for the chi-square distribution with 3 degrees of freedom are 7.815 and 11.345, respectively. Of the 48 chi-square values in Table 3, 3 (indicated with *) exceed 7.815 and none exceeds 11.345, which shows our sequence passes the collision test.

In the OPSO test [18], we consider a two-dimensional unit cube and divide it into $2^{10} \times 2^{10}$ cells (cubes) with equal volumes. Using $n = 2^{21}$ random numbers x_t $(1 \leqslant t \leqslant n)$, we distribute n points whose coordinates are given by (x_t, x_{t+1}) with the exception of the last point determined by (x_n, x_1). The number of empty cells should be approximately normally distributed with mean 141 909 and standard deviation 290.26 if the sequence $\langle x_t \rangle$ is perfectly random.

For each initial value IX we generated $50n$ random numbers and performed the above test 50 times. The normalized numbers of empty cells are shown in Table 4; asterisks (*) indicate the deviates exceeding in absolute value 5% significant level 1.96. In order to test whether or not the 50 deviates for each IX can be regarded as a random sample from the standard normal distribution, we computed a K–S statistics K_{50}^+ and K_{50}^- which are shown at the bottom of the table. This shows that our sequence passes the OPSO test which Marsaglia [18] claims is stringent.

6. Concluding remarks

It has been shown that good pseudorandom sequences can be generated by the linear recursion modulo two. The crucial point is to use a primitive polynomial of sufficiently large degree and to initialize the sequence so that the independence of bits is guaranteed. The sequence

Table 4
OPSO test

IX	1985	9158	9851	8915	8519	5891	5198	1589
1	0.686	−1.413	2.336*	0.568	−0.283	1.240	−0.596	−1.147
2	0.396	0.630	0.427	−1.288	0.765	0.754	2.098*	−0.469
3	−0.196	−0.803	−2.512*	−0.455	0.003	0.975	−0.262	−0.320
4	0.803	0.052	0.954	−1.230	0.906	1.130	−0.493	−0.706
5	0.410	−0.834	0.872	−2.394*	−0.072	0.679	2.446*	0.041
6	−0.775	0.810	−0.792	0.103	0.799	−0.723	−1.833	0.682
7	−1.130	2.753*	−1.874	0.513	1.278	−1.292	0.345	0.641
8	−0.758	−1.388	−0.789	0.007	−0.165	−0.482	−1.044	−1.719
9	0.388	−1.257	1.581	0.658	1.192	−0.245	−0.110	−0.448
10	0.017	−1.850	−1.333	0.524	0.396	−0.606	0.734	−1.864
11	1.340	−0.496	1.557	−1.426	0.589	1.354	0.562	−1.878
12	0.465	−0.589	−0.076	0.079	−0.872	0.500	−0.493	1.526
13	−1.413	−0.045	0.000	−0.217	0.689	2.160*	−0.848	0.017
14	0.562	−1.082	−0.096	−1.292	0.179	−1.382	−1.454	−0.338
15	0.875	0.686	−0.158	−0.978	0.589	0.424	0.531	−0.630
16	−1.964*	−0.565	1.437	−1.609	−0.899	−0.806	−0.462	−0.179
17	−0.761	1.330	−0.682	0.307	1.957	1.071	0.424	−0.341
18	−0.706	−0.203	−0.121	−0.131	−0.534	−0.754	−0.544	0.710
19	−1.864	1.075	−0.420	−0.586	−0.096	−0.338	0.620	2.257*
20	0.913	−0.513	0.003	0.448	−1.023	−0.751	−0.606	−3.686*
21	−0.813	0.386	0.982	−1.395	−2.029*	−0.007	−1.288	0.238
22	−1.106	0.768	−1.523	0.283	0.217	0.407	0.286	0.627
23	1.051	−1.464	0.024	0.686	−0.572	0.699	−1.096	1.664
24	−0.059	1.202	1.326	0.265	−1.078	−2.064*	1.316	−0.968
25	−0.014	−0.644	−0.782	0.079	0.555	0.548	0.675	0.648
26	0.289	−0.806	−0.479	1.368	0.138	0.637	−0.479	0.713
27	0.152	−1.199	−1.402	−0.417	−1.585	−1.257	0.727	−1.233
28	1.009	1.382	−1.147	0.103	−0.220	0.289	−2.050*	−0.493
29	−0.944	−0.579	0.603	1.313	0.269	0.717	−0.358	−0.637
30	1.561	−0.692	1.275	−0.558	0.451	0.748	1.457	−0.220
31	−0.861	0.737	0.562	1.223	0.114	1.137	1.767	0.651
32	−0.110	1.282	−0.317	1.633	−0.320	0.427	0.713	0.713
33	−0.200	−0.389	0.193	0.169	1.333	−0.482	2.053*	−0.314
34	0.348	−0.124	−0.262	0.772	0.189	0.513	0.183	−0.944
35	0.806	0.696	−1.354	−1.113	0.648	1.020	1.481	−0.768
36	0.916	1.282	1.475	1.030	−0.258	−1.506	0.052	1.399
37	−0.734	−0.245	2.219*	−0.420	−0.217	0.438	0.823	−0.241
38	−1.464	0.003	−0.034	2.591*	−1.464	−0.179	0.792	−2.215*
39	−0.410	0.079	0.183	−1.854	−0.703	0.689	−0.479	0.076
40	0.114	0.572	1.636	−0.100	0.544	1.943	0.289	−0.796
41	0.348	0.565	−0.978	−0.944	2.102*	−1.075	0.448	0.475
42	−1.382	1.144	0.407	−0.744	−0.520	0.162	1.626	−1.967*
43	1.068	0.475	1.878	−1.368	−0.231	0.283	−0.451	0.844
44	1.705	0.989	2.866*	1.030	−0.131	−0.906	−0.593	−0.723
45	0.837	−0.923	0.737	1.137	0.465	−0.127	0.599	0.689
46	0.513	0.417	−1.044	−0.703	0.675	−0.334	1.061	0.413
47	−1.247	−1.261	−0.617	0.283	−1.178	−1.995*	1.612	0.537
48	−0.396	0.127	1.454	0.582	−2.188*	−1.578	0.103	0.238
49	−1.275	−1.543	−0.820	−0.431	1.078	−0.596	0.668	−0.358
50	0.517	−0.875	−2.146*	−0.338	−0.393	−0.903	−0.572	1.185
K_{50}^{+}	0.707	0.920	0.445	0.499	0.510	0.339	0.241	0.834
K_{50}^{-}	0.538	0.292	0.981	0.223	0.667	0.833	0.937	0.245

generated by the program in the Appendix is 16-distributed and has virtually no autocorrelation for lags up to about $2^{521}/32 \approx 2 \times 10^{155}$.

Quite contrary to our result, Marsaglia [18] reports very bad results on sequences generated by recurrences modulo two. He used primitive polynomials of degrees 17, 31 and 55, which, in opinion of the present writer, are too small for the sequences to be random. It is desirable that the sequence is at least 10-distributed, which means we must choose the degree of the primitive polynomial at least as large as 10 times the bit length of the sequence.

Acknowledgement

Main parts of this paper are based on two previous papers [7,22] written in Japanese. The key idea of the generator was proposed by T. Saito, an ex-student of the present writer, who also coded the original version of the program in the Appendix.

Appendix. A FORTRAN program

```
00000001 C*****************************************
00000002 C M-SEQUENCE BY A PRIMITIVE POLYNOMIAL
00000003 C WITH MANY TERMS
00000004 C  CODED BY T. SAITO
00000005 C  REVISED BY M. FUSHIMI
00000006 C*****************************************
00000007 C------- INITIALIZE -------
00000008       SUBROUTINE INTLZ(IX)
00000009       INTEGER P,P2,P3,Q
00000010       PARAMETER (P=521,P2=P*2,P3=P*3,Q=32)
00000011       INTEGER  SRC,DST,W(P3),X(P3),X0(P2),
00000012      +  X1(P2),X2(P2),X3(P2),BIT(32),WI
00000013       COMMON //SRC,DST,W
00000014       DO 5 I=1,P2
00000015       X0(I)=0
00000016       X3(I)=0
00000017     5 CONTINUE
00000018       BIT(32)=0
00000019       BIT(1)=2**30
00000020       DO 10 J=2,31
00000021         BIT(J)=BIT(J-1)/2
00000022    10 CONTINUE
00000023       IR=IX
00000024       DO 20 I=1,P
00000025         IR=IR*69069
00000026         IF(IR.GT.0) X0(I)=1
00000027    20 CONTINUE
00000028       DO 25 I=P+1,P2
00000029         X0(I)=MOD(X0(I-P)+X0(I-Q),2)
00000030    25 CONTINUE
00000031       X3(2)=1
00000032       DO 30 I=1,P-1
00000033         IMD2=MOD(I,2)
00000034         DO 31 J=P,1,-1
00000035           X3(2*J-IMD2)=X3(J)
00000036           X3(2*J-1+IMD2)=0
00000037    31   CONTINUE
00000038         CALL MODP(X3,P,Q)
00000039    30 CONTINUE
00000040       CALL PRODCT(X0,X3,X1,P)
00000041       DO 35 I=P+1,P2
00000042         X1(I)=MOD(X1(I-P)+X1(I-Q),2)
00000043    35 CONTINUE
00000044       CALL PRODCT(X1,X3,X2,P)
00000045       DO 40 I=1,P
00000046         X(3*I-2)=X0(I)
00000047         X(3*I-1)=X1(I)
00000048         X(3*I)=X2(I)
00000049    40 CONTINUE
00000050       DST=1
00000051       SRC=3*(P-Q)+1
00000052       DO 50 I=1,P3
00000053         WI=0
00000054         DO 51 J=1,32
00000055           IF(X(DST).EQ.0) GO TO 53
00000056           WI=WI+BIT(J)
00000057    53     X(DST)=MOD(X(DST)+X(SRC),2)
00000058           DST=MOD(DST,P3)+1
00000059           SRC=MOD(SRC,P3)+1
00000060    51   CONTINUE
00000061         W(I)=WI
00000062    50 CONTINUE
00000063       RETURN
00000064       END
00000065 C ------- INNER PRODUCT ------
00000066       SUBROUTINE PRODCT(F,G,H,P)
00000067       INTEGER F(1),G(1),H(1),P,WORK
00000068       DO 10 I=1,P
00000069        WORK=0
00000070        DO 20 J=1,P
00000071         IF(G(J).EQ.1) WORK=WORK+F(I+J-1)
00000072    20  CONTINUE
00000073        H(I)=MOD(WORK,2)
00000074    10 CONTINUE
00000075       RETURN
00000076       END
00000077 C ------- RESIDUAL POLYNOMIAL ------
00000078       SUBROUTINE MODP(H,P,Q)
00000079       INTEGER H(1),P,Q,R
00000080       R=P-Q
00000081       DO 10 I=P,1,-1
00000082         H(I)=MOD(H(I)+H(I+P),2)
00000083         H(I+R)=MOD(H(I+R)+H(I+P),2)
00000084         H(I+P)=0
00000085    10 CONTINUE
00000086       RETURN
00000087       END
00000088 C --- GENERATE THE NEXT RANDOM NUMBER ---
00000089       SUBROUTINE RND(IRND,FRND)
00000090       INTEGER P,Q,P3
00000091       PARAMETER (P=521,Q=32,P3=P*3)
00000092       INTEGER SRC,DST,W(P3)
00000093       COMMON // SRC,DST,W
00000094       IRND=W(DST)
00000095       FRND=IRND*0.4656613E-9
00000096       W(DST)=IEOR(W(DST),W(SRC))
00000097       DST=MOD(DST,P3)+1
00000098       SRC=MOD(SRC,P3)+1
00000099       RETURN
00000100       END
```

References

[1] A.C. Arvillias and D.G. Maritsas, Partitioning the period of m-sequences and application to pseudorandom number generation, *J. Assoc. Comput. Mach.* **25** (1978) 675–686.

[2] H. Bright and R. Enison, Quasi-random number sequences from a long TLP generator with remarks on application to cryptography, *Computing Surveys* **11** (1979) 357–370.

[3] J.M. Chambers, *Computational Methods for Data Analysis* (Wiley, New York, 1977).

[4] B.J. Collings and G.B. Hembree, Initializing generalized feedback shift register pseudorandom number generators, *J. Assoc. Comput. Mach.* **33** (1986) 706–711.

[5] G.S. Fishman, *Principles of Discrete Event Simulation* (Wiley, New York, 1978).

[6] M. Fushimi, A reciprocity theorem on the random number generation based on m-sequences and its applications, *Trans. Inform. Process. Soc. Japan* **24** (1983) 576–579 (in Japanese).

[7] M. Fushimi, Statistical tests of eight hundred million pseudorandom numbers based on an M-sequence, *Japan. J. Appl. Statist.* **15** (1986) 147–162 (in Japanese).

[8] M. Fushimi, On misunderstandings about pseudorandom number generation by linear recurrence modulo two, in: *First IASC (Internat. Assoc. Statist. Computing) World Conf. on Computational Statistics and Data Analysis*, Shizuoka, Japan, 1987.

[9] M. Fushimi, Statistical tests of pseudorandom numbers generated by a linear recurrence modulo two, in: *First APORS (Assoc. Asian-Pacific Oper. Res. Soc. IFORS) Conf.*, Seoul, Korea, 1988.

[10] M. Fushimi and S. Tezuka, The k-distribution of the generalized feedback shift register pseudorandom numbers, *Comm. ACM* **26** (1983) 516–523.

[11] S.W. Golomb, *Shift Register Sequences* (Aegean Park Press, Laguna Hills, CA, 1982).

[12] T. Imai and M. Fushimi, Pseudorandom number generators whose subsequences are multidimensionally equidistributed, *Trans. Inform. Process. Soc. Japan* **26** (1985) 454–458 (in Japanese).

[13] W.J. Kennedy and J.E. Gentle, *Statistical Computing* (Marcel Dekker, New York, 1980).

[14] S. Kirkpatrick and E.P. Stoll, A very fast shift-register sequence random number generator, *J. Comput. Phys.* **40** (1981) 517–526.

[15] D.E. Knuth, *The Art of Computer Programming, Vol. 2, Seminumerical Algorithms* (Addison-Wesley, Reading, MA, 2nd ed., 1981).

[16] T.G. Lewis and W.H. Payne, Generalized feedback shift register pseudorandom number algorithms, *J. Assoc. Comput. Mach.* **21** (1973) 456–468.

[17] G. Marsaglia, Random numbers fall mainly in the planes, *Proc. Nat. Acad. Sci. U.S.A.* **61** (1968) 25–28.

[18] G. Marsaglia, A current view of random number generators, in: L. Billard, Ed., *Computer Science and Statistics: Proc. 16th Symp. on the Interface* (North-Holland, Amsterdam, 1985) 3–10.

[19] G. Marsaglia and L.-H. Tsay, Matrices and the structure of random sequences, *Linear Algebra Appl.* **67** (1985) 147–156.

[20] W.H. Payne and K.L. McMillen, Orderly enumeration of nonsingular binary matrices applied to text encryption, *Comm. ACM* **21** (1978) 259–263.

[21] A. Ralston and C.L. Meek, Eds., *Encyclopedia of Computer Science* (Petrocelli/Charter, New York, 1976).

[22] T. Saito, M. Fushimi and T. Imai, High-speed M-sequence random number generation based on the primitive polynomials with many terms, *Trans. Inform. Process. Soc. Japan* **26** (1985) 148–152 (in Japanese).

[23] R.C. Tausworthe, Random numbers generated by linear recurrence modulo two, *Math. Comp.* **19** (1965) 201–209.

[24] J.P.R. Tootill, W.D. Robinson and A.G. Adams, The runs up-and-down performance of Tausworthe pseudo-random number generators, *J. Assoc. Comput. Mach.* **18** (1971) 381–399.

[25] J.P.R. Tootill, W.D. Robinson and D.J. Eagle, An asymptotically random Tausworthe sequence, *J. Assoc. Comput. Mach.* **20** (1973) 469–481.

[26] N. Zierler and J. Brillhart, On primitive trinomials (mod 2) II, *Inform. and Contr.* **14** (1969) 566–569.